TP3

Constructeur par recopie

déclare ⇒ construit

`V3 v1(v2);`

`v1` construit par le constr. par copie (copie les val. de v_2)

dict. .h

arg V3 — copie...

this

`V3(const V3 &v)`

constructeur V3 ← qui prend en arg un V3

on déclare / construit un nouvel obj en init sa val par copie de v_2

Affectation

=

`V3 v1 = v2;`

① constructeur par défaut pour v_1

② affectation $v_1 \leftarrow v_2$

un objet déjà existant / construit (v_1)

modifie sa valeur.

`V3 v1 = 4.;`

type V3

V3 ? → constructeur `V3(float x=0, float y=0, float z=0)` opt.

`V3(4.)`

`V3 & operator=(const V3 & v2)`

`v1 = (v2 = v3);`

retour ref v_2

`i = j = k = 3;`

Vector — ⚠

↳ stdlib c++ ⇝ vector — super-tableaux !

`t[i]` → cases ....

⊕ ⟶ accès à la taille
⟶ méthodes de réallocation ↗ agrandir
↘ diminuer.

↓

gestion mémoire.

C++ ⟶ allocation mém : new → retourne un pointeur ..... ⚠ tableau ex: tab de 10 int

⟶ désallocation : delete + delete[] → tableaux

```
int *t =
new int[10];
```

: détruire un tab : delete[] t

Vector ⟶ constructeur : • prend une taille → stocke ds _taille
• alloue un tableau de taille

⟶ destructeur → désalloue le tableau.

Partage des résultats du sondage

Les participants sont maintenant en train de consulter les résu

1. Etant donnée une classe :

```
class Toto {
public:
  Toto(int i) { _data = new float[i] ; _taille = i ; }
private:
  float * _data ;
  int _taille ;
} ;
```

_taille existe

Le destructeur fait delete _data ; delete _taille ;	(0) 0%
Le destructeur fait delete _data ;	(1) 11%
Le destructeur fait delete[] _data ;	(5) 56%
Le destructeur fait delete[] _data ; delete _taille ;	(3) 33%

_taille : entier ← ~~new~~

↓

construit automatiquement juste avant la construct. de Vector.

↳ détruite automatiquement. ~~delete~~

Partage des résultats du sondage

Les participants sont maintenant en train de consulter les résul

1. Etant donnée une classe :

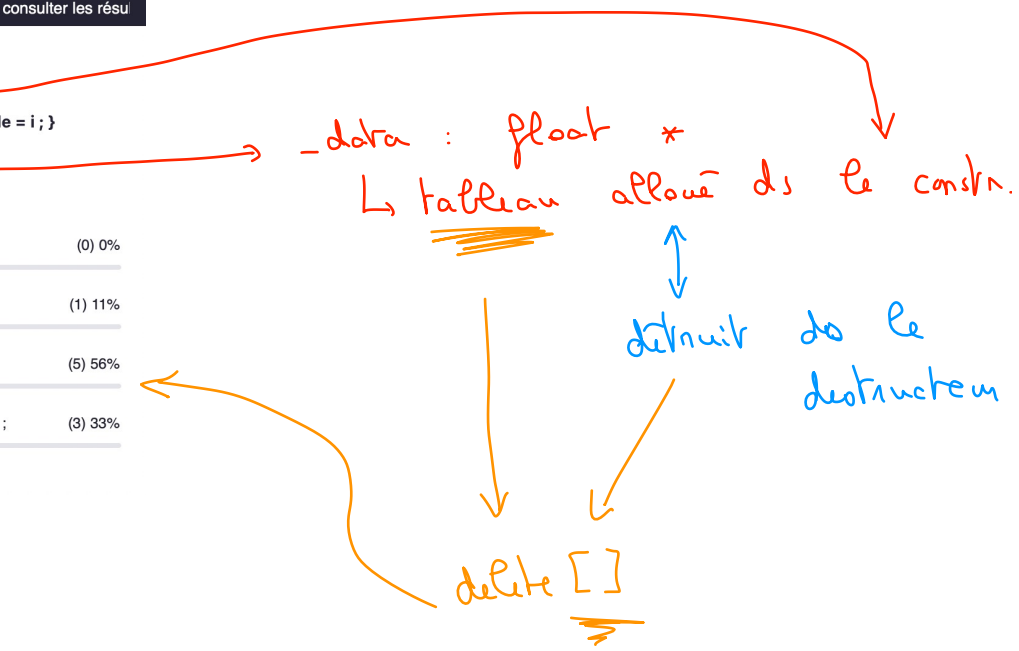
```
class Toto {
public:
  Toto(int i) { _data = new float[i] ; _taille = i ; }
private:
  float * _data ;
  int _taille ;
} ;
```

Le destructeur fait delete _data ; delete _taille ; (0) 0%

Le destructeur fait delete _data ; (1) 11%

Le destructeur fait delete[] _data ; (5) 56%

Le destructeur fait delete[] _data ; delete _taille ; (3) 33%



Const. défaut ⇝ init "de base" / logique.

↳ V3 ⇝ (0,0,0)

↳ Vecteur

{

tableau

↳ taille par défaut → pratique

stable : 128

↳ pas de réalloc systématique.

main :

Vecteur t ;

↓

défaut

crée un tab ....

constructeur :

↳ construit

① Vecteur ( )

{ _data = new int[128] ; ← @ d'un tableau ← alloué

_taille = 128 ;

}

pointeur

) destructeur

delete []

② → Vecteur(int taille) = 128 ....

Vecteur(const Vecteur & v)

→ alloue _data de taille v._taille
→ init _taille
→ for i .....
_data[i] = v._data[i]

---

static → attaché à la classe (pas à 1 objet)

class Toto {
private:
int _mbr;

1 var _mbr par objet

static int _mbr2;

main: Toto t1, t2;

t1._mbr t2._mbr

public:
① int get_mbr();
② ~~int get_mbr2();~~
↳ ③ static int get_mbr2(); variable de la classe.

meth. statique

① t1.get_mbr();
t2.get_mbr();

② t1.get_mbr2();
t2.get_mbr2();

~~objet~~

accède à la même var (classe)

static : → lié à la classe
→ variables
→ méthodes

③ Vecteur::get_mbr2()

Init des données membres statiques → ~~constructeur~~

↘ au début du .cpp. ⌒ Vector.cpp.

↓

déclaration + init

↙

```
int Vector::_nb2 = 4;
```